Task sequence optimization for a dual-robot assembly system using evolutionary algorithms

J-H Park* and S-H Ryu
School of Mechanical Engineering, Hanyang University, Seoul, Korea

Abstract: For the purpose of improving assembly task productivity, a multirobot assembly system has been introduced. In this system, multirobots operate simultaneously and transfer parts from the part feeders to the printed circuit boards (PCBs) moving at a constant speed on the conveyor line. To exploit the capacity of this system to the full, the system should be set up with an optimal conveyor line speed and each robot should be operated along a proper assembly sequence. In this paper, a dual-robot assembly system is considered, and an efficient algorithm for concurrent optimization of conveyor line speed and assembly sequence is developed. Various evolution algorithms are tried as an optimization method, and, based on the results, the optimal conveyor line speed and assembly sequence are obtained. When using the optimized conveyor line speed and assembly sequence, the operation time can be reduced by up to 22.8 per cent compared with the initial set-up.

Keywords: dual-robot assembly system, conveyor line speed, assembly sequence, evolutionary algorithm

NOTATION

d	distance in the x coordinate
D_b	width of each board
m^A, m^B	board assembly sequence by robot A and B
M	number of workpieces (boards) on the conveyor line
n^A, n^B	part assembly sequence by robot A and B
N	number of parts assembled on a single board by each robot
q_f	joint angle of a robot at the final position
q_s	joint angle of a robot at the starting position
q_1	first joint angle of a robot
q_2	second joint angle of a robot
$T_{b,travel}$	time required for backward robot motion
$T_{p,travel}$	time required for forward robot motion
T_{pick}	time required for picking up a part
T_{place}	time required for placing and assembling a part
T_i^l	time for ith operation by robot l
v_c	conveyor speed
x_f	final position
x_s	starting position

The MS was received on 31 March 2000 and was accepted after revision for publication on 15 September 2000.
**Corresponding author: School of Mechanical Engineering, Hanyang University, Hangdang, Sungdong, Seoul, Korea.*

1 INTRODUCTION

In a manufacturing process that assembles a single product out of many parts, the assembly operation time directly affects productivity. Therefore, the operation time needs to be reduced for an improvement in productivity. For this purpose, Li *et al.* [1] suggested a new type of assembly system that consists of multirobots and a conveyor line. In this system, high-speed multirobots are distributed along a precision conveyor line and transfer parts simultaneously from the part feeders to the printed circuit boards (PCBs) moving at a constant speed on the conveyor line.

The throughput is ultimately decided by the conveyor line speed in this system. However, if the speed is too high, the assembly system cannot accomplish all the tasks before the PCBs move into the next position. Therefore, the conveyor line speed should be decided on the basis of the assembly time. The assembly time varies according to the conveyor line speed as well as the assembly sequence of each robot. Therefore, the conveyor line speed and assembly sequence must be optimized simultaneously to increase the system throughput.

For systems of this kind the hardware configuration is rather simple but the sequence planning is quite complicated because each robot should avoid mutual geometric and dynamic interference during the simultaneous robotic operations. This kind of planning has already

been tried in the past using a simulated annealing method for the dual-robot system [2]. The simulated annealing method is known to be efficient for the problems of combinatorial optimizations, but the global optimum point was hard to obtain owing to the nature of the dual-robot assembly problem which includes geometric constraints and interference. In this paper, evolutionary algorithms were employed which is more efficient for finding the global optimum.

Evolutionary algorithms are stochastic optimization techniques that simulate the natural evolutionary process. Chromosome representation is specified and gene operators applied on it are devised. Finally, from the optimization results the optimal conveyor line speed and assembly sequence are suggested for each robot, and the optimization efficiency of each algorithm is compared.

2 PROBLEM STATEMENT

2.1 Robot assembly task representation

The aforementioned dual-robot assembly system consists of double two-link robots, part feeders and a moving conveyor line as shown in Fig. 1. All the parts are stored in several part feeders and taken by the end-effector of the robot. If it is assumed that $2N$ parts should be mounted to make a complete product, then each robot will assemble N parts. For convenience, A and B are used to denote the spaces approachable by robot A and robot B, as shown in Fig. 2. The identification numbers of the part feeders reached by robots A and B at the ith sequence will be denoted by n_i^{A} and n_i^{B} respectively. Then the assembly sequences of robots A and B at the side of the part feeder can be represented as follows:

$$n^{\mathrm{A}} = (n_1^{\mathrm{A}}, n_2^{\mathrm{A}}, \ldots, n_N^{\mathrm{A}}) \tag{1}$$

$$n^{\mathrm{B}} = (n_1^{\mathrm{B}}, n_2^{\mathrm{B}}, \ldots, n_N^{\mathrm{B}}) \tag{2}$$

where $n_i^{\mathrm{A}} \in A$ and $n_i^{\mathrm{B}} \in B$, $1 \leqslant i \leqslant N$.

After picking up a part, each robot should mount it on one of the PCBs being transferred on the conveyor line. If it is assumed that M boards are approachable by robots at any instant, then similar notations can be used to designate the sequence of the board approached by robots A and B. The identification numbers of the board reached by robots A and B at the ith sequence

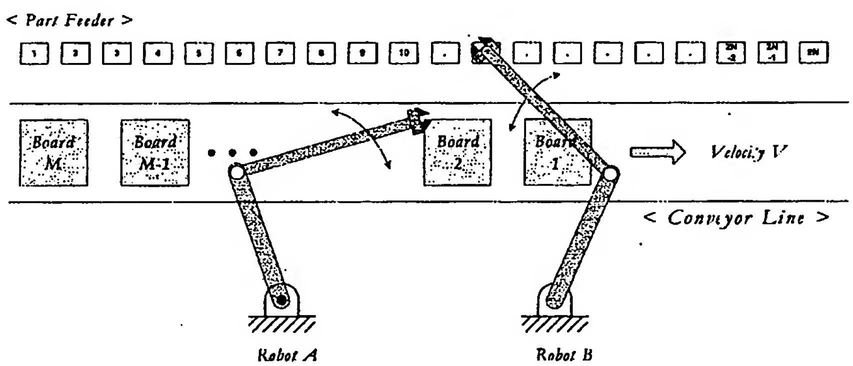

Fig. 1 Structure of a dual-robot system

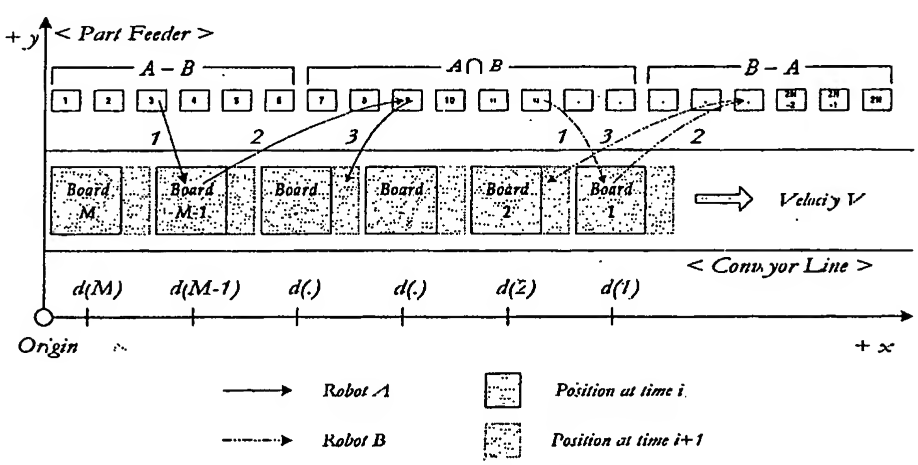

Fig. 2 Coordinated motion of the two robots

will be denoted by m_i^A and m_i^B respectively. Then the assembly sequences of robots A and B at the side of the conveyor line can be represented as follows:

$$m^A = (m_1^A, m_2^A, \ldots, m_N^A) \tag{3}$$

$$m^B = (m_1^B, m_2^B, \ldots, m_N^B) \tag{4}$$

where $m_i^A, m_i^B \in \{1, 2, \ldots, m\}$.

2.2 Robot motion constraints

While two robots accomplish the assembly task concurrently, an impulsion or dynamic interference may occur between them. Dynamic interference in this case is a phenomenon where the degree of assembly precision becomes worse because of unwanted vibrations or reaction forces caused by the robot motions in high acceleration. In this paper, to simplify the problem, it is assumed that the two robots can escape from mutual impulsion or interference by keeping some safety distance and by moving at the same pace.

Consider the case where the hands of the two robots are located at the side of the part feeder. Here, the part feeders can be reached by both robots or by only one robot depending on their positions. It is assumed that the two robots cannot enter $A \cap B$ at the same time, i.e. if $n_i^A \in A \cap B$, then $n_i^B \notin A \cap B$ and vice versa, as shown in Fig. 2. The robots always keep some safety distance from each other at the side of the part feeders.

At the side of the conveyor line, the constraint that the two robots should keep a minimum safety distance during assembly task performance is imposed. The constraint is expressed by the following equation:

$$d(m_i^A) + d_{\min} < d(m_i^B) \tag{5}$$

where $d(m_i^A)$ is the x coordinate of the end-effector of robot A when it is on board m_i^A, $d(m_i^B)$ is the x coordinate of the end-effector of robot B when it is on board m_i^B and $d_{\min}$ is the minimum safety distance.

In order to prevent dynamic interference between the robots, another constraint, that the two robots should move and assemble at the same pace, is imposed on the system. The robot that arrives first above a target board should wait there until the other robot also reaches its target board without starting any assembly task. This avoids a situation where one robot is moving at high speed while the other robot is performing a precise assembly task, and as a result dynamic interference can be eliminated.

2.3 Definition of cost function

Figure 2 shows the movements of the two robots during an assembly task. Each robot picks up a part from the part feeder and mounts it into one board. Next, it moves back to the part feeder and repeats the same operations. The object of optimization is to reduce the total working time including the time for robot motions and for the assembly job. Here, the working time of a robot at the ith assembly step is calculated as follows:

$$T_i^l = T_{\text{p,travel}_i}^l + T_{\text{pick}} + T_{\text{b,travel}_i}^l + T_{\text{place}} \tag{6}$$

where T_{pick} is the time needed to pick up a part, T_{place} is the time needed to mount a part, $T_{\text{p,travel}_i}^l$ is the moving time of robot l from the previous board to a new part feeder and $T_{\text{b,travel}_i}^l$ is the moving time of robot l from the chosen part feeder to a new board on which robot l will mount the selected part. On the other hand, the time needed for the ith assembly step is decided by equation (7) considering the dynamic constraint

$$T_i = \max(T_i^A, T_i^B) \tag{7}$$

where T_i^A and T_i^B are the working time of robots A and B at the ith assembly step respectively.

The cost function is defined as the total working time for the dual-robot system to complete the whole assembly task for a single complete product according to the planned sequence as follows:

$$f_c = \sum_{i=1}^{N} T_i \tag{8}$$

To calculate T_i^l, the values of T_{pick}, T_{place}, $T_{\text{p,travel}_i}^l$ and $T_{\text{b,travel}_i}^l$ should be known. For convenience, it is assumed that T_{pick} and T_{place} are constant values. The time consumption for the movement of robot l, T_{travel}^l, is decided by the location of the start point and end point of robot movement and the characteristic of the robot arm. However, if the robot arm structure and controller are given, then two ending points are the only factors to decide the time consumption during movement.

Consider the case where robot l moves from the jth part position to the kth board position. If the jth part position is denoted as $x_p(j)$ and the kth board position is denoted as $x_b(k)$ and it is assumed that at the ith step a part is taken from feeder j and placed on board k, then the moving time of robot l is a function of the two coordinates as follows:

$$\begin{aligned} T_{\text{travel}_i}^l &= f[(x_p(j), x_b(k)] \\ &= f[(x_p(n_i^l), x_b(m_i^l)] \end{aligned} \tag{9}$$

Here, function f depends on the robot and the control system.

In order to estimate the robot motion time, the start point and end point of robot movement will be denoted as x_s and x_f respectively. Then the Cartesian coordinates of the terminal points, x_s and x_f, should be transformed to joint coordinates q_s and q_f using the inverse kinematic relation. Then each joint has to move $|q_{1s} - q_{1f}|$ and $|q_{2s} - q_{2f}|$ respectively. For a two-link robot, all the

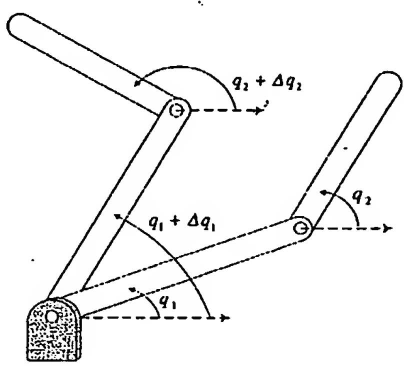

Fig. 3 Minimum number of coordinates to display robot movement

possible robot link movements can be represented using only three variables, $q_1 - q_2$, Δq_1 and Δq_2 as shown in Fig. 3, because of the symmetry. For given terminal points, the robot motion time can be computed from numerical simulations. The dual-robot system was modelled on computer, and a table indicating the robot motion time for each case was compiled (Table 1). Since the number of combinations for the start and end positions is too large, use is made of an interpolation method from the computer simulation data, and they are used for calculating the objective function in the process of optimization.

3 OPTIMIZATION USING EVOLUTIONARY ALGORITHMS

The above optimization problem is quite hard to solve by the conventional optimization technique because of its complexity. In this paper, evolutionary algorithms are used. These are stochastic optimization techniques that simulate the natural evolutionary process. They can often outperform conventional optimization methods when applied to difficult real-world problems. Here, a few different kinds of evolutionary algorithm will be applied to our problem and compared on the basis of the results, and optimal solutions will be obtained.

Generally, evolutionary algorithms have similar procedures:

Table 1 Data table for robot motion time

$q_1 - q_2$ (initial joint angle difference) (rad)	Δq_1 (difference in joint angle 1) (rad)	Δq_2 (difference in joint angle 2) (rad)	t (robot moving time) (s)
—	—	—	—
1.7	−0.9	−0.7	0.4365
1.7	−0.9	−0.6	0.4431
—	—	—	—

(a) displaying optimization solution candidates as chromosomes;
(b) trial of gene operation (mutation, crossover, etc.) to each chromosome;
(c) choosing the superior individuals as the next generation using some methods (the fittest survival theory);
(d) finding the optimal solution as the generation is repeated.

These methods do not require much mathematical background for optimization, and an arbitrary object function and constraints can be chosen. Also, the solution search area is so broad that the probability of finding the global minimum is very high, and it is very easy to combine with an existing simple classical optimization method such as a simple downhill method.

3.1 Encoding problem

To apply an evolutionary algorithm to an optimization problem, a solution of the problem should be encoded into a chromosome. Traditionally, a chromosome is a simple binary string, but this is not well suited to a combinatorial optimization problem such as that examined here. Therefore, the permutation representation method will be used [3, 4]. This is perhaps the most natural representation of the path of a robot, where assembly parts and PCBs are listed in the order in which they are visited respectively. An example of the representation is as follows:

Chromosome of robot A about part assembly order:

[5 5 6 6 5 6 4 6 5 4]

Chromosome of robot A about board assembly order:

[1 3 12 2 6 8 7 13 4 5]

Chromosome of robot B about part assembly order:

[3 1 2 1 3 1 2 5 3 5]

Chromosome of robot B about board assembly order:

[19 15 17 9 20 11 16 14 18 10]

3.2 Validity of chromosome

Classical gene manipulation methods cannot be applied directly to the above chromosomes. For example, if the traditional one-point crossover operator is used, then the above permutation representation may lead to illegal tours of robots. Consider the process of making an offspring from the parent chromosomes of robot A as shown in Fig. 4. If use is made only of the traditional one-point crossover operation, then the offspring chromosome cannot include parts 3 and 4, though the parent chromosomes select all parts 1 to 4 once.

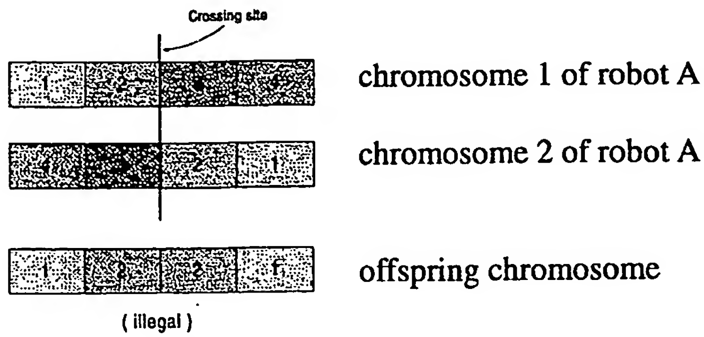

Fig. 4 Conventional crossover method

Another point that needs to be considered is that the distance constraints between robot A and B should be kept even after the gene operation is completed. Otherwise an impulsion or dynamic interference between robots may be induced in the offspring generation. Therefore, gene operation methods ensuring the reproduction of new chromosomes that allow robots to keep a proper distance from each other are required. Here, the notion of the valid chromosome is introduced.

Definition 1

The chromosome for an assembly task is valid if the resulting operation of the system is as follows:

1. Each assembly part is taken and mounted on a board just once by one of the two robots.
2. The two robots keep the safety distance from each other whenever they pick or mount a part respectively.

Based on the above definition, mutation and crossover operators that always produce valid chromosomes will be defined in the following sections.

3.3 Mutation operation

Among many existing mutation methods, the reciprocal exchange mutation method [5] is appropriate for the present problem. It selects two positions at random and then swaps the numbers of the chromosomes on these positions. However, for the present problem the two robots should be considered simultaneously because of the safety distance constraints. Therefore, 'simultaneous reciprocal exchange mutation operation' was introduced, as shown in Fig. 5. This method applies the reciprocal exchange mutation method to both chromosomes of robots A and B at the same time. If the initial sequence planning is done correctly, then offspring chromosomes created by this method remain valid even after mutation operation.

$$\begin{array}{l} \text{Part selection order of A } [\,4 \quad \boxed{6} \quad 10 \quad \boxed{1} \quad \ldots \,] \\ \text{Part selection order of B } [\,13 \quad \boxed{20} \quad 18 \quad \boxed{13} \quad \ldots \,] \end{array}$$

Fig. 5 Simultaneous reciprocal exchange mutation operation

3.4 Crossover operation

As a crossover operation method, the 'position-based crossover' method proposed by Syswerda [6] is appropriate for the present problem. However, as in the above mutation operation, the movement of the two robots needs to be considered concurrently. Therefore, the 'two-phase position-based simultaneous crossover' method is suggested as shown in Fig. 6. In this method, firstly a crossover operation of the parent chromosomes of robot A is done as follows. A computer selects a random number of positions from one parent chromosome and produces a protochild by copying the numbers on these positions into the corresponding positions of the protochild. Next, it deletes the numbers already selected from the second parent chromosome and then the remaining sequence of numbers contains all the numbers the protochild needs to be a valid chromosome. Then the remaining numbers are placed into the unfixed positions of the protochild from left to right according to the order of the sequence to produce an offspring chromosome of robot A. For robot B, an offspring chromosome is created by making the same position changes to one parent chromosome of robot B as from the first chromosome of robot A to the offspring chromosome.

4 OPTIMIZATION OF CONVEYOR LINE SPEED AND ASSEMBLY SEQUENCE

4.1 Development of a simultaneous optimization algorithm

The throughput of the assembly system is ultimately dependent on the conveyor line speed. To fully exploit the capacity of this system, the conveyor line speed

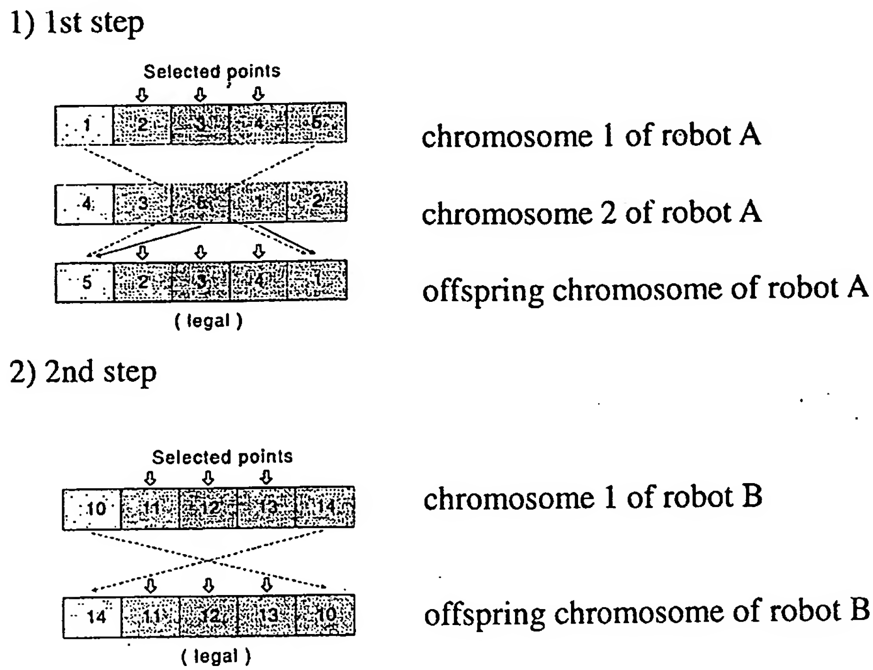

Fig. 6 Two-phase position-based simultaneous crossover method

must be increased as much as possible. However, an upper limit for the speed should be determined by the fact that each robot must be able to finish its own assembly task before the PCBs move to the next position.

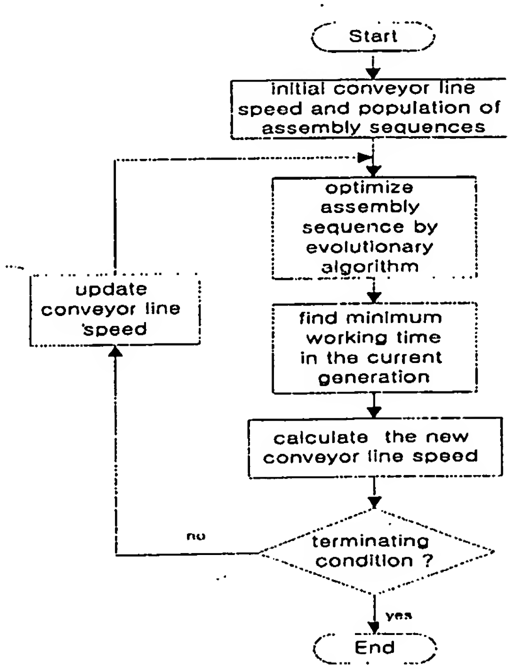

Fig. 7 Flow chart of simultaneous optimization using an evolutionary algorithm

Therefore, firstly the working time is reduced as much as possible by optimizing the assembly sequence at an initial constant conveyor line speed, and from the obtained minimum working time the new conveyor line speed is calculated. By the way, if the conveyor line speed is changed, then the working time changes even for the same assembly sequence. Therefore, they should be optimized simultaneously, and for that purpose an algorithm was developed. Its flow chart is shown in Fig. 7.

According to the flow chart, after each generation the computer updates the conveyor line speed on the basis of the minimum working time in that generation. In this process, the following equation is used:

$$v_c(t+1) = \frac{D_b}{t_w(t)} \tag{10}$$

where v_c is the speed of the conveyor line, D_b is the width of each board, t_w is the minimum working time in the current generation for the system to make a unit product and t represents the generation.

If both conveyor line speed and working time are optimized, then it is possible to set up the system with the optimal conveyor line speed and operate each robot along the optimal sequence.

4.2 Optimization using evolutionary algorithms

In this paper, four kinds of evolutionary algorithm [7–9] were tried to solve the optimization problem, and in this section it is explained how they were applied. In the case

Table 2 Optimization parameters for evolutionary algorithms

	Algorithm			
Content	Evolution strategy	Genetic algorithm	Breeder genetic algorithm	Parallel genetic algorithm
Population size	10	10	10	10
Generations	60	60	60	60
Mutation probability	0.7	0.7	0.7	0.7
Crossover probability	—	0.5	0.5	0.5
Selection strategies	Truncation selection	Proportionate selection	Truncation selection	Truncation selection
Local hill-climbing operations	—	—	Simple downhill method	Simple downhill method

considered, 20 parts are required to be mounted to make a single product and six PCBs are always approachable by the two robots. In this case, each robot should mount 10 parts into each PCB. The optimization parameters for each algorithm are shown in Table 2.

4.2.1 The $(\mu + \lambda)$ evolution strategy

An evolution strategy is a random search that uses only mutation and selection. Generally, this algorithm is superior to other evolutionary algorithms in the sense of convergence speed and calculation time. The algorithm procedure is as follows:

Step 1. Create an initial population of size λ.
Step 2. Compute the fitness $F(x_i)$, $i = 1, \ldots, \lambda$.
Step 3. Select the $\mu < \lambda$ best individuals.
Step 4. Create λ/μ offspring of each of the μ individuals by mutation.
Step 5. If not finished, return to Step 2.

Here, the values of λ and μ were set up as 10 and 5 respectively. As a mutation method, the simultaneous reciprocal exchange mutation shown in Fig. 5 was used. To allow many perturbations to each chromosome, the probability of mutation was set as 0.7. As a result, an optimum conveyor line speed and assembly sequence were searched and this reduced the assembly working time by about 16.5 per cent compared with an arbitrary assembly schedule and stationary conveyor line. The CPU time for optimization was taken to be about 39.2 s per generation.

4.2.2 Genetic algorithm

The genetic algorithm is the most widely known type of evolutionary algorithm. Here it was applied according to the following procedure:

Step 0. Define a genetic representation of the problem.
Step 1. Create an initial population $P(0) = x_1^0, \ldots, x_N^0$.
Step 2. Compute the average fitness $\bar{F} = \sum_{i=1}^{N} F(x_i)/N$. Assign each individual the normalized fitness value $F(x_i^t)/\bar{F}$.
Step 3. Assign each x_i a probability $p(x_i, t)$ proportional to its normalized fitness. Using this distribution, select N vectors from $P(t)$. This gives the set $S(t)$.
Step 4. Pair all of the vectors in $S(t)$ at random, forming $N/2$ pairs. Apply crossover with probability p_{cross} to each pair and other genetic operators such as mutation, forming a new population $P(t+1)$.
Step 5. Set $t = t + 1$, return to Step 2.

Here, the population size was set as 10 and a proportionate selection method was used in constituting the next generation. As a mutation method, the simultaneous reciprocal exchange mutation operation shown in Fig. 5 was used, and as a crossover method, the two-phase position-based simultaneous crossover method shown in Fig. 6 was used. In addition, the probability of mutation and crossover were set as 0.7 and 0.5 respectively. As a result, an optimum conveyor line speed and assembly sequence were searched and this reduced the assembly working time by about 20.7 per cent compared with an arbitrary assembly schedule and stationary conveyor line. The CPU time for optimization was taken to be about 52.2 s per generation.

4.2.3 Breeder genetic algorithm

The major difference between the genetic algorithm and the breeder genetic algorithm is the method of selection. Here, use was made of the truncation selection method, and therefore the T per cent best individuals of a population were selected as parents for the constitution of the next generation. This algorithm was applied according to the following procedure:

Step 0. Define a genetic representation of the problem.
Step 1. Create an initial population $P(0)$.
Step 2. Each individual performs local hill climbing.
Step 3. The breeder selects T per cent of the population for mating. This gives set $S(t)$.
Step 4. Pair all the vectors in $S(t)$ at random, forming N pairs. Apply the genetic operators (crossover and mutation), forming a new population $P(t+1)$.
Step 5. Set $t = t + 1$, return to Step 2 if it is better than some criterion (acceptance).
Step 6. If not finished, return to Step 3.

Here the population size and T value were set as 10 and 50 per cent respectively. Therefore, the computer chooses the five best individuals of the total population in the

current generation and obtains the other five by applying gene operations to those chosen chromosomes. Here, the proportionate selection method which controls the evolution centrally is not used and instead each individual performs local hill climbing using a simple downhill optimization procedure during its lifetime. The procedure for the simple downhill method is as follows. Apply the aforementioned simultaneous reciprocal exchange mutation operation randomly to each chromosome and then take the perturbed chromosome only for the case where the assembly working time is decreased. In this case, because only a decrease in assembly working time is allowed, a simple downhill optimization is achieved. The probability of mutation and crossover were set as 0.7 and 0.5 respectively, and the gene operations used were the aforementioned simultaneous reciprocal exchange mutation operation and two-phase position-based simultaneous crossover method. As a result, an optimum conveyor line speed and assembly sequence were searched and this reduced the assembly working time by about 21.4 per cent compared with an arbitrary assembly schedule and stationary conveyor line. The CPU time for optimization was taken to be about 89.8 s per generation.

4.2.4 *Parallel genetic algorithm*

A genetic algorithm is a parallel random search with centralized control. Here, the centralized part is the selection schedule and this requires the calculation of the average fitness of the population which takes much time. On the other hand, a parallel genetic algorithm does not have any centralized control. This makes each individual perform local hill climbing and select a partner for mating in only its neighbourhood. When the offspring is generated, it also performs local hill climbing and replaces the parent if it is better than some criterion (acceptance). In this algorithm, individuals evolve in parallel and the information exchange within the whole population is a diffusion process because the neighbourhoods of the individuals overlap. All decisions are made by the individuals themselves. Therefore, the PGA is a totally distributed algorithm without any central control. This models the natural evolution process which self-organizes itself. Here, this was applied according to the following procedures:

Step 0. Define a genetic representation of the problem.
Step 1. Create an initial population and its population structure.
Step 2. Each individual performs local hill climbing.
Step 3. Each individual selects a partner for mating in its neighbourhood.
Step 4. An offspring is created with genetic operators working on the genotypes of its parents.
Step 5. The offspring performs local hill climbing. It replaces the parent if it is better than some criterion (acceptance).
Step 6. If not finished, return to Step 3.

Here, use was made of the aforementioned simple downhill method for the local hill-climbing procedure. The probability of mutation and crossover were set as 0.7 and 0.5 respectively and the same gene operations as in the past section were used for both mutation and crossover. As an optimization result, an optimum conveyor line speed and assembly sequence were searched and this reduced the assembly working time by about 22.8 per cent with an arbitrary assembly schedule and stationary conveyor line. The CPU time for optimization was taken to be about 64.5 s per generation. Owing to the time consumption in the local hill-climbing procedures, the effect of calculation time decrease for each generation by the deletion of selection procedure was not pronounced. However, more perturbations of chromosomes were accomplished within a generation, and this reduced the convergence time to 193.5 s, as shown in Table 3.

4.3 Comparison of optimization results

The optimization results using four evolutionary algorithms (evolution strategy, genetic algorithm, breeder genetic algorithm, parallel genetic algorithm) are displayed in Figs 8 and 9. As the generation is repeated, both the assembly working time and the conveyor line speed converged to constant values. In these figures, each evolutionary algorithm searched a different optimal solution at a different generation. The efficiency of an algorithm can be inspected from the convergence speed and the obtained optimization results. The CPU time taken for each optimization method is listed in Table 3. Here, the evolution strategy algorithm takes the shortest

Table 3 CPU time for optimization using evolutionary algorithms

	Algorithm			
Content	Evolution strategy	Genetic algorithm	Breeder genetic algorithm	Parallel genetic algorithm
Total run time (s)	2352	3132	5388	3870
Time/generation (s)	39.2	52.2	89.8	64.5
Convergence time (s)	274.4	1722.6	1616.4	193.5
(generation)	(7)	(33)	(18)	(3)

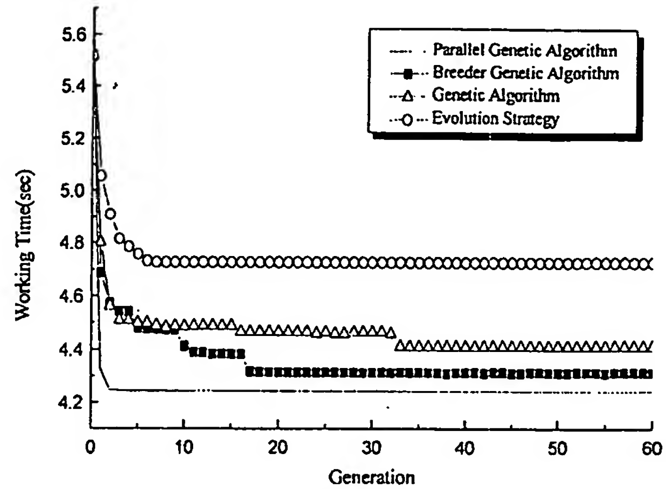

Fig. 8 Working time history during simultaneous optimization using evolutionary algorithms

time per generation and the parallel genetic algorithm spends the smallest number of generation iterations before converging. This was possible because it made full use of the simple downhill method during each generation lifetime and passed down the obtained properties to the next generation. The searched optimal results, assembly working time and conveyor line speed are listed in Tables 4 to 6. Among the four algorithms, the parallel genetic algorithm could search the highest optimal conveyor line speed and the smallest assembly working time. Considering this algorithm is also good in regard to convergence speed, as mentioned before, it can be concluded that this is the most appropriate algorithm for the present problem.

To justify the comparison, each algorithm was tried five times and the results are displayed in Tables 4 and 5. The results were used to compile Table 3 which shows the average optimization results of each algorithm. From this, the reduced amount of assembly working time of each algorithm can be compared. It can be seen that, in the optimization using the parallel genetic algorithm, an assembly time reduction of about 22.8 per cent was possible in comparison with the case of a stationary conveyor line and arbitrary assembly sequence planning, and this was the nearest value to the global minimum working time. The initial and optimal robot assembly sequences are displayed as follows:

Initial part selection order of robot A:

[10 3 2 4 1 5 13 7 11 6]

Initial part selection order of robot B:

[20 8 9 14 12 19 15 17 16 18]

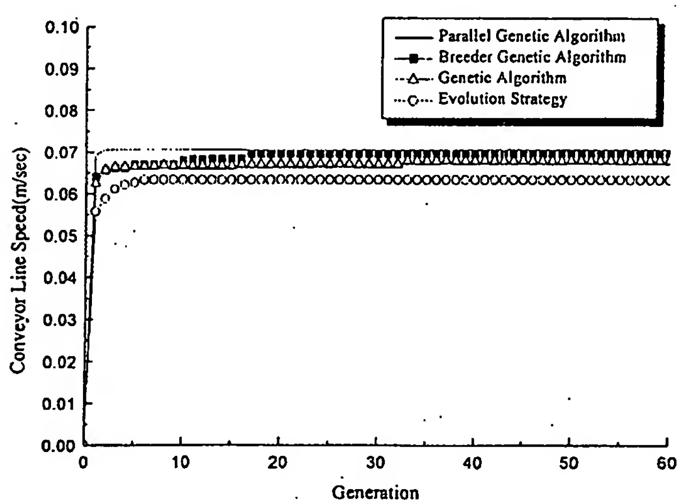

Fig. 9 Conveyor line speed history during simultaneous optimization using evolutionary algorithms

Table 4 Optimal assembly time using evolutionary algorithms

	Algorithm			
Content	Evolution strategy	Genetic algorithm	Breeder genetic algorithm	Parallel genetic algorithm
1	4.52	4.41	4.31	4.25
2	4.73	4.35	4.37	4.30
3	4.60	4.36	4.31	4.24
4	4.67	4.41	4.33	4.25
5	4.54	4.39	4.36	4.28
Average (s/board)	4.61	4.38	4.34	4.26

Table 5 Optimal conveyor line speed using evolutionary algorithms

	Algorithm			
Content	Evolution strategy	Genetic algorithm	Breeder genetic algorithm	Parallel genetic algorithm
1	0.0664	0.0680	0.0696	0.0706
2	0.0634	0.0690	0.0686	0.0698
3	0.0652	0.0688	0.0696	0.0708
4	0.0642	0.0680	0.0693	0.0706
5	0.0661	0.0683	0.0688	0.0701
Average (m/s)	0.0651	0.0685	0.0691	0.0704

Table 6 Optimization results using evolutionary algorithms

	Algorithm				
Content	Random initial value	Evolution strategy	Genetic algorithm	Breeder genetic algorithm	Parallel genetic algorithm
Average optimal working time (s/board)	5.52	4.61	4.38	4.34	4.26
Working time reduction (%)		16.5	20.7	21.4	22.8

Optimized part selection order of robot A:

[4 6 9 13 2 8 3 5 1 7]

Optimized part selection order of robot B:

[20 14 15 16 10 18 11 12 19 17]

Initial part selection order of robot A:

[5 4 1 5 3 1 2 5 1 2]

Initial part selection order of robot B:

[6 5 4 6 6 2 5 6 4 6]

Optimized part selection order of robot A:

[3 5 1 4 4 3 1 3 5 3]

Optimized part selection order of robot B:

[5 6 6 6 5 6 2 5 6 6]

The time spent for each assembly step is displayed in Figs 10 and 11 for an arbitrary and an optimal assembly set-up respectively. From the figures it can be seen that the optimized operation yields synchronized movements of robot A and robot B so that the wasted time at each step is reduced. As a result, the total working time can be minimized.

5 CONCLUSIONS

In this paper, a dual-robot assembly system has been considered. In this system two robots operate simultaneously and transfer parts from the part feeders to the PCBs moving at a constant speed on the conveyer line. An efficient algorithm has been developed to optimize the conveyor line speed and robot assembly sequence simultaneously to decrease the assembly working time as much as possible. As an optimization method, use was

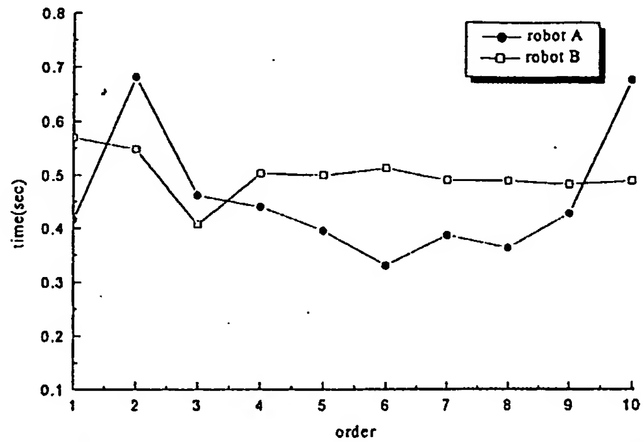

Fig. 10 Working time at each step with an arbitrary assembly sequence and conveyor line speed

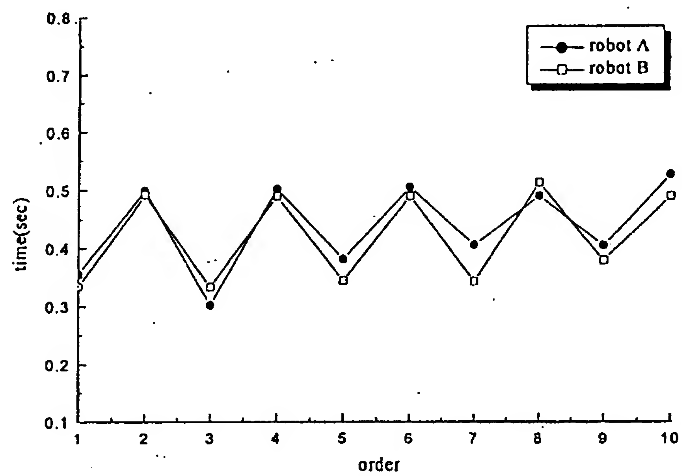

Fig. 11 Working time at each step with an optimal assembly sequence and conveyor line speed

made of evolutionary algorithms (evolution strategy, genetic algorithm, breeder genetic algorithm, parallel genetic algorithm) and in this process a permutation encoding method was introduced and appropriate gene manipulation methods were devised.

As an optimization result, an optimal conveyor line speed and robot movement sequence for the assembly were searched, allowing the user fully to exploit the dual-robot assembly system without inducing any geometric or dynamical interference between robots. If this dual-robot system is implemented, the optimal conveyor line speed is set and each robot is operated along the optimal sequence, and PCB productivity can be increased by reducing the assembly working time maximum by 22.8 per cent compared with an arbitrary assembly operation. On the other hand, according to optimization results, the parallel genetic algorithm was most efficient in solving the present problem considering both the convergence speed and optimization results. When a multiple robot assembly system is implemented to increase the productivity further, the conveyor line speed and robot motion sequences could be decided through optimization using the evolutionary algorithms suggested in this paper.

REFERENCES

1 **Li, S.-H., Fujiwara, N.** and **Asada, H.** An ultrahigh speed assembly robot system: Part II: design. In Proceedings of Japan–USA Symposium on *Flexible Automation*, 1994.
2 **Park, J.-H.** and **Huh, K.** Optimal task planning for high speed robotic assembly using simulated annealing. *Jap. Soc. Mech. Engrs Int. J., Ser. C*, 2000, 43(1), 222–229.
3 **Davis, L.** (Ed.) *Handbook of Genetic Algorithms*, 1991 (Van Nostrand Reinhold, New York).
4 **Michalewicz, Z.** *Genetic Algorithm + Data Structure = Evolution Programs*, 2nd edition, 1994 (Springer-Verlag, New York).

5 Alender, J. Interval arithmetic and genetic algorithms in global optimization. In *Artificial Neural Nets and Genetic Algorithms*, 1995, pp. 387–391 (Springer-Verlag, New York).
6 Davis, L. Applying adaptive algorithms to domains. In Proceedings of International Joint Conference on *Artificial Intelligence*, 1985, pp. 162–164.
7 Banzhaf, W. and Eeckman, F. H. *Evolution and Biocomputation*, 1995 (Springer-Verlag, Berlin and Heidelberg).
8 Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1989 (Addison-Wesley).
9 Gen, M. and Cheng, R. *Genetic Algorithms and Engineering Design*, 1997 (John Wiley).